Addendum to AVEC Report:
NGGPS Level-1 Software Evaluation

Advanced Computing Evaluation Committee

Chair: John Michalakes, NOAA (IMSG)
Co-chair: Mark Govett, NOAA/ESRL
Rusty Benson, NOAA/GFDL
Tom Black, NOAA/EMC
Henry Juang, NOAA/EMC
Alex Reinecke, NRL
Bill Skamarock, NCAR

Contributors

Michael Duda, NCAR
Thomas Henderson, NOAA/ESRL (CIRA)
Paul Madden, NOAA/ESRL (CIRES)
George Mozdzynski, ECMWF
Ratko Vasic, NOAA/EMC

Final report submitted: 28 May 2015

I. Introduction

On April 30th, 2015, the Advanced Computing Evaluation Committee (AVEC) submitted the computational performance and scalability results from Level-1 benchmarks of five Next Generation Global Prediction System (NGGPS) candidate models. This addendum provides a preliminary evaluation of the candidate model software as described in Section VI of the AVEC report and as called for in the Next Generation Global Prediction System (NGGPS) Dynamic Core Testing Plan (Section V). The software evaluation is intended to highlight strengths and identify potential weaknesses with respect to maintainability, extensibility, development process, and performance portability of the candidate software packages in their current states, which are assumed to be preliminary. The evaluations involve review of self-reports by the candidate model groups in the form of responses to a questionnaire[1] (completed), follow-up interviews and inspection of code (in progress), and review of documentation (not yet begun). The following interim evaluation is provided at this time to coincide with release of the completed parts of the Phase-I Test Report to the Dynamic Core Test Group. Committee members Benson, Black, Michalakes, Reinecke, and Skamarock approve this report addendum; one committee member, Mark Govett, has indicated his dissent.

[1] See "NGGPS Level-1 Software Evaluation Criteria and Procedures", AVEC report to NGGPS program, December 11, 2014.

In general, we found nothing in the software design, implementation, or practices to invalidate any candidate model outright. Different models are at different levels of maturity. The size and makeup of the developer groups and level of support they provide also vary, but a more detailed evaluation will be needed to estimate the effort and resources required to bring a given model to a state of readiness for NGGPS.

Table 1 provides a summary of the modeling group responses to the software evaluation questionnaire. The candidate cores are listed as column headings: FV3 (GFDL), NMM-UJ (NCEP), MPAS (NCAR), NIM (ESRL), and NEPTUNE (NRL). ECMWF provided responses for the IFS model to the software evaluation questionnaire which were reviewed by AVEC but are not included in the summary, since IFS is not an NGGPS candidate. The following paragraphs provide additional explanation.

Programming language. All of the models are written in standard Fortran. The FV3 model is able to make use of "Co-Arrays", a feature in the Fortran 2008 language standard for exploiting very-low latency one-sided message-passing where supported (e.g. Cray systems). Some of the models (FV3, MPAS, and NEPTUNE) have additional lower level or library infrastructure support that is written in another language such as C. This practice is widely accepted within the NWP software development community.

Parallelism: Message Passing. All of the models use standard Message Passing Interface (MPI) mechanisms for coarse-grain distributed-memory parallel decomposition and computation of their forecast domain over multiple processes for running on large HPC clusters. While the grids and mechanical details of task-parallelism differ, all of the candidate models exploit parallelism in the two horizontal dimensions of their forecast domain, and all implement a "nearest-neighbor" pattern of message passing that allows weak scaling to arbitrarily large grid sizes and processor counts.

Parallelism: Thread-parallelism. Threads provide medium-grain parallelism over multiple processor cores that share access to memory on a node. Threads may also provide concurrency within a single core, which may allow the code to better tolerate memory access latency. Both uses are important for efficiency on next-generation processor architectures (MIC and GPU) as well as current and future generations of conventional processors containing large numbers of processor cores.

FV3, NIM, and NMM-UJ implement thread-parallelism at the level of individual loops: that is, different iterations of a loop that has been annotated with an OpenMP directive run in parallel on different threads that synchronize at the beginning and end of the loop. Neither MPAS nor NEPTUNE currently support thread parallelism; they rely on MPI-parallelism alone. Both groups responded that OpenMP thread-parallelism is planned.

In FV3 and NMM-UJ, the threaded dimension may vary in different sections of the code – thread-parallel loops may be over the vertical dimension in some places and over a horizontal dimension in others. Depending on computer architecture and the access and reuse patterns of a computation, switching between thread-parallel dimensions may cause unnecessary data movement between caches or NUMA[2] regions. NIM consistently threads loops over only one (horizontal) domain dimension.

Storage/Loop nesting Order:

The choice of inner-loop and innermost (fastest running) array indices will have implications for optimizing for fine-grained (vector or SIMT[3]) performance on both novel (GPU and MIC) and conventional processor architectures. The appropriateness of a storage/loop nesting order may vary between different phases of the computation (dynamics versus physics) or between different discretizations. Table 1 lists the choices for storage/loop-nesting order reported by the modeling groups.

Extensible Software Design. A well-architected design that manages tradeoffs between performance and flexibility is crucial for developing, maintaining, extending and supporting NWP model software across a range of current and future platforms to the community of users, from research to operations. Fully understanding and managing the design and implementation of an NGGPS candidate will be ongoing and demanding. The evaluation here is preliminary.

FV3, NIM and MPAS are rigorously architected with ESMF-conforming Init/Run/Finalize calling interfaces for major structural components and are readily adaptable to NEMS. FV3 conforms to and runs within GFDL's Flexible Modeling System (FMS), an ESMF precursor. NIM's sister model FIM is already part of the NEMS/ESMF framework at NCEP. NMM-B, NCEP's earlier NGGPS candidate, is in NEMS and features the most developed moving nest capability of the candidate models. However, the NMM-UJ code that replaced NMM-B in the NGGPS testing has a flat program structure indicating a quickly constructed prototype. The NEPTUNE model, also relatively new, has a similar flat main program structure but, like the NMM-UJ, is consistently modular underneath.

Nesting or Mesh Refinement. Models of performance requirements for global NWP indicate that very high resolution (3 km and higher) simulations will be *possible* on next generation HPC systems, but very expensive in terms of resources, including electricity.[4] From a cost point of

[2] Non-uniform memory access, a type of memory system architecture employed on many systems with multi-processor nodes.

[3] Single Instruction, Multiple Thread, which is fine-grained parallelism over GPU threads (distinct from medium-grained parallel OpenMP threads).

[4] Michalakes, J. NOAA Operational Forecasting and the HPC Imperative. 16th ECMWF Workshop ojn HPC in Meteorology. Reading, U.K. October, 2014. http://www.ecmwf.int/sites/default/files/HPC-WS-Michalakes.pdf

view, uniform very high resolution deterministic forecasts over a global domain may never be feasible for day-to-day operational forecasting, even with exascale computers. Therefore, a model's ability to focus resolution over a region of interest through *in-place mesh refinement* or *nesting* is likely to be an NGGPS requirement for the foreseeable future. In-place mesh refinement involves a single mesh that has increased numbers of smaller grid cells over an area of refinement. The refinement may be static or adaptive. A nest is a separate higher resolution mesh that receives forcing information from a lower resolution parent domain (1-way interaction) and that may also feed information back to the parent (2-way interaction). The location of a nest may be fixed within the parent domain or it may move to follow of a feature of interest in the simulation (e.g. a hurricane).

MPAS and NEPTUNE provide in-place mesh refinement. MPAS implements static refinement with no plans to implement adaptive or moving areas of refinement. NEPTUNE plans to have adaptive mesh refinement. FV3 currently supports refinement via static, nested regional grids and stretched grids, with plans to provide movable nests that are restricted to be wholly contained within the boundaries of one face of the cubed-sphere. EMC plans to transfer moving nest mechanisms currently implemented in NMM-B to NMM-UJ. NIM has plans to provide 1- or 2-way interactive nesting capability.

Coding practices. Fortran provides a number of standard language features that allow the programmer to make assertions about the code that are automatically checked for correctness when the code is compiled. The IMPLICIT NONE statement allows the compiler to check for misspelled, mistyped or otherwise undeclared variables. The INTENT attribute allows the compiler to check whether an argument to a subroutine or function has been incorrectly accessed or assigned. Argument keywords provide another mechanism for argument agreement checking. With regard to physics interfaces, adherence to the Kalnay et al.[5,6] rules helps make physics more interoperable between models. All of the models use IMPLICIT NONE statements and INTENT attributes. FV3 uses argument keywords consistently; the other models use argument keywords to some extent. FV3, NMM-UJ, and NIM use Kalnay-conforming physics interfaces.

Reproducibility. Two forms of reproducibility were evaluated: the ability to give the same results of a simulation running on different numbers of processors and the ability to give the same result for a run that is stopped and then restarted. FV3 and NIM provide both bit-for-bit restarts and bit-for-bit results on different numbers of processors. NIM also has demonstrated bit-for-bit reproducibility between the CPU, MIC and GPU for its dynamical core. NMM-UJ

[5] E. Kalnay, M. Kanamitsu, J. Pfaendtner, J. Sela, J. Stackpole, J. Tuccillo, M. Suarez, L. Umscheid, and D. Williamson, 1989: Rules for Interchange of Physical Parameterizations. *Bull. Amer. Meteor. Soc.*, **70**, 620–622. doi: http://dx.doi.org/10.1175/1520-0477(1989)070<0620:RFIOPP>2.0.CO;2

[6] Doyle, et al., 2015. Revisiting Kalnay's "Rules for Physics Interoperability" 25 years later." Presentation to Eugenia Kalnay Symposium, Amer. Met. Soc. Annual Meeting. Phoenix, Arizona. https://ams.confex.com/ams/95Annual/webprogram/Paper260152.html

provides bit-for-bit results on different numbers of processors and bit-for-bit restarts are planned. MPAS provides bit-for-bit restarts; bit-for-bit agreement of results on different numbers of processors is planned but not yet implemented in MPAS. NEPTUNE provides bit-for-bit restarts but does not provide bit-for-bit agreement of results on different numbers of processors.

Advanced Architectures: Only the NIM model is capable of running on GPU and MIC. Other models have advanced architectures in their plans.

Table 1: Summary of Responses to Software Evaluation Questionnaires

	FV3	NMMUJ	MPAS	NIM	NEPTUNE
Languages	Fortran, Co-Array Fortran, and C	Fortran	Fortran and C	Fortran	Fortran and C
MPI	2D horz. decomposition and over cube faces; Nearest neighbor	2D horz. decomposition and over cube faces; Nearest neighbor	2D horz. decomposition; Nearest neighbor	2D horz. decomposition; Nearest neighbor; Specified with SMS (ESRL tool) directives	2D horz. decomposition; Nearest neighbor
OpenMP	Loop-level, threaded dimension varies; subroutine level around physics	Mostly loop-level, threaded dimension varies; Subroutine level over radiation	Planned (1-2 years); Subroutine level	Loop level, threading is consistently over horizontal dimension; subroutine level around physics	Planned
Storage/Loop Nesting Order	IJK	IJK	K-innermost	K-innermost	K-innermost
Extensible software design	Abstraction layers and APIs for domain definition & mgmt., comms, I/O, etc.; Supports Init/Run/Finalize component interfaces; Uses FMS Framework	Use of Fortran MODULES; Flat program structure (everthing called from main); NEMS/ESMF (planned)	Abstraction layers and APIs for domain definition & mgmt., comms, I/O, etc.; Supports Init/Run/Finalize component interfaces	Abstraction layers and APIs for domain definition & mgmt., comms, I/O, etc.; Supports Init/Run/Finalize component interfaces	Use of Fortran MODULES (I/O, domain decomposition, physics, dynamics right hand side, dynamics time integration, etc.). Flat program structure.
Nesting/Refinement	Nesting within a cubed-sphere face or stretched grid; 1-way and 2-way; static (non-moving); plans for moving nests	Plans for moving 1- and 2-way interacting nests; no geographic restrictions on movement	In-place refinement (inherently 2-way); static (non-moving) refinement	Plans for 1- and 2-way interacting nests; static (non-moving)	In-place refinement (inherently 2-way); adaptive mesh refinement planned; no cost estimate
Coding practices	IMPLICIT NONE; INTENT(IN/OUT/INOUT); Argument keywords used; Kalnay-conforming physics APIs	IMPLICIT NONE; INTENT(IN/OUT/INOUT); Some argument keywords; Kalnay-conforming physics APIs	IMPLICIT NONE; INTENT(IN/OUT/INOUT); Some argument keywords	IMPLICIT NONE; INTENT(IN/OUT/INOUT); Argument keywords used; Kalnay-conforming physics APIs	IMPLICIT NONE; INTENT(IN/OUT/INOUT); Some argument keywords
Reproduciblity	Bit-for-bit reproducible on different core counts; Bit-for-bit restarts	Bit-for-bit reproducible on different core counts; Bit-for-bit restarts (planned)	Not bit-reproducible on different numbers of MPI tasks (but under development); Bit-for-bit restarts	Bit-for-bit reproducible on different core counts; Bit-for-bit restarts	Not bit-reproducible on different numbers of MPI tasks; Bit-for-bit restarts
Current and Advanced Architectures	Originally developed for vector systems; has adapted to each new architectural paradigm and now developed and supported on conventional multi-core processors. Hybrid MPI/OpenMP makes well suited for MIC; exploring GPU with directives-based or CUDA programming models.	Developed and supported on conventional multi-core processors; RRTMG physics has been adapted to accelerators (MIC and GPU).	Developed and supported on conventional multi-core processors. No efforts currently for novel architectures.	Developed and supported as single-source on conventional multi-core, MIC and GPU.	Developed and supported on conventional multi-core processors. Testing kernels on MIC and GPU.